Challenge SSTIC 2015

Vincent Hamon

I. Stage 1

A. sdcard.img

A l'aide de file, et strings on voit qu'on est sur du ducky rubber.

L'image est du fat16

> sudo mount -r -o loop,fat=16 ./sdcard.img ./test

→ on récupère inject.bin

B. inject.bin

un projet ducky-decode-read-only (http://ducky-decode.googlecode.com/svn) permet de décoder inject.bin

il s'agit d'un script qui lance powershell et écrit un fichier.

Avec quelques lignes imparfaites de python on obtient stage2.zip

II. Stage 2

Le début des soucis pour ceux qui sont trop mauvais pour tricher.

J'ai tenté le reverse de la map mais devant le nombre d'image qu'ils avaient mis et qui étaient toutes référencées j'ai abandonné et passé quelques heures à apprendre le rocker jump.

Le pire étant qu'une fois que j'ai réussi à arriver dans la salle qui donne les infos sur la clé à utiliser, j'ai pris une photo sans la vérifier : le flash la faisait trop briller pour que j'arrive à distinguer ce qu'il y avait dessus.→ tout à refaire…

bref on obtient une clé : "9e2f31f78153296B3D9B0BA67695DC7CB0DAF152B54CDC34FFE0D35526609FAC".

L'algo et l'iv étant fournis on obtient le stage3

III. Stage 3

A. trace cap.

Avec wireshark on voit qu'il s'agit de la trace d'une souris USB.

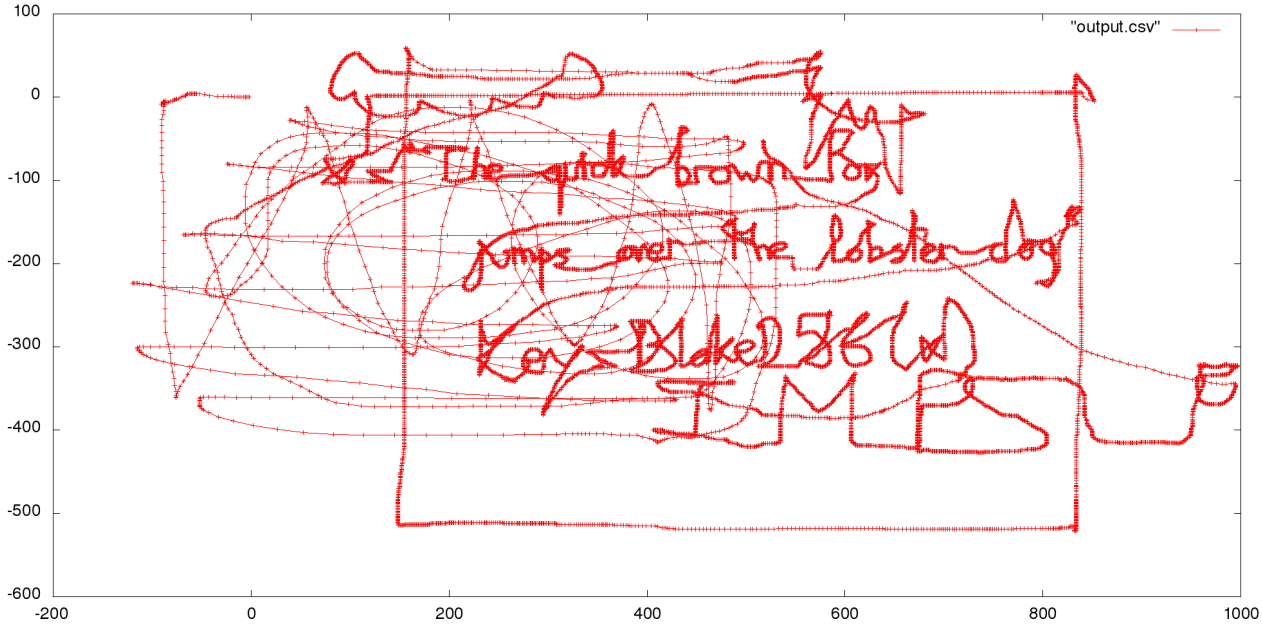
D'après la spec du device, la position de la souris est donnée en relatif.

On filtre usb.capdata et on se génère un fichier qui est une succession des positions relatives de la

souris.

On parse ce fichier avec quelques lignes de python et on génère un fichier csv avec la position absolue de la souris.

On obtient :



B. Blake256

On trouve assez facilement sur le web un fichier blake.py qui implémente l'algo de hash qu'il nous faut.

C. Serpent CTS

Pour Serpent c'est plus compliqué. Autant on trouve des implémentations de Serpent dans les « modes classiques » autant en CTS c'est moins simple.

2 alternatives :

- utiliser ECB et implémenter CTS par dessus
- trouver une implé de Serpent en CTS

J'ai choisi (fini par choisir) la deuxième et ai pris cryptopp.

Après avoir fait toutes les erreurs du monde (mauvais mot de passe Maj/min à certains endroits, mauvais copier/coller CTR au lieu de CTS…) dans les 10 lignes de c++ et les 63 de python, j'ai fini par obtenir le hash demandé.

IV. Stage 4

A. Stage4.html

- Des données

- un hash
- un unpacker javascript obfusqué

B. désobfuscation

J'ai utilisé un plugin firefox javascript deobfuscator.

Il permet de retourner une version quasi lisible des fonctions lorsque celles-ci sont appelées. Je crois avoir utilisé un autre framework de deobfuscation pour refaire du formatage mais je ne le retrouve plus.

On obtient ça :

```
__ = document;
$ = "<b>Failed to load stage5</b>";
___ = 'write';
____ = 'getElementById';
_$_ = "raw";
$$ = window;
__$ = $$.crypto.subtle;
__$_ = 'decrypt';
___$ = 'status';
$____ = "importKey";
_______ = 0;
__$__ = 'then';
_$____ = 'digest';
__$___ = 'innerHTML';
___$__ = {name:'SHA-1'};
____$_ = data;
_____$ = hash;
_$_____ = Blob;
___$___ = URL;
____$__ = 'createObjectURL';
______$ = 'type';
$_______ = 'application/octet-stream';
_$______ = 'name';
__$_____ = 'AES-CBC';
___$____ = 'iv';
____$___ = "<a href=\"";
_____$__ = "\" download=\"stage5.zip\">download stage5</a>";
______$_ = '(';
_______$ = ')';
_________ = parseInt;
__________ = $$['navigator']["userAgent"];
_____________ = 'length';
______________ = 'substr';
_______________ = 1;
________________ = 2;
_________________ = 8;
__________________ = 16;
___________________ = 'indexOf';
____________________ = 'charCodeAt';
```

```
_____ = 'push';
_____ = Uint8Array;
_____ = '';
_____ = 'byteLength';
_____ = "toString";
__['write']('<h' + 1 + '>Down' + 'load' + ' ' + 'manager</h' + 1 + '>');
__['write']("<div id=\"status\"><i>loading...</i></div>");
__['write']("<div style=\"display:none\"><a target=\"blank\" href=\"chrome://browser/content/preferences/preferences.xul\">Back to preferences</a></div>");
function _____(_____){
  _ = [];
    for(_____ = _____; _____ < _____[_____]; ++_____)
_[_____](_____[_____](_____));
  return new _____(_);
}
function _____(_____){
  _ = [];
   for(_____ = _____; _____ < _____[_____] / _____; ++_____)  _[_____](_____(_____[_____](_____ * _____, _____), _____));
  return new _____(_);
}
function _____(_____){
  _____ = _____;
    for(_____ = _____; _____ < _____[_____]; ++_____) {
    ___ = _____[_____][_____](_____);
    if(___[_____] < _____) _____ += _____;
    _____ += ___;
  }
  return _____;
}
function _____(){
    $_ = _____(_____[_____](_____[_____](_____$_) + _____, _____));
   _$__ = _____(_____[_____](_____[_____](_____$) - _____, _____));
  _$ = { };
  _$[_$_____] = __$_____;
  _$[___$____] = $_;
  _$[_____] = _$__[_____] * _____;
  __$[$____](_$_, _$__, _$, false, [__$_])[__$__](function(_$___) {
    __$[__$_](_$, _$___, _____(____$_))[__$__](function(___$_) {
      ____$ = new _____(___$_);
      __$[_$____](___$__, ____$)[__$__](function(____$$) {
        if(_____$ == _____(new _____(____$$))) {
          _____$_ = { };
          _____$_[_____$] = $_____;
          _____$ = new _$_____([____$], _____$_);
          __$____ = ___$___[____$__](_____$);
          __[____](___$)[__$___] = ____$___ + __$____ + _____$__;
        } else {
```

```
          __[_____](___$)[__$___] = $;
        }
      });
    }).catch(function() {
      __[_____](___$)[__$___] = $;
    });
  }).catch(function() {
    __[_____](___$)[__$___] = $;
  });
}
$$['setTimeout'](_____, 1000);
```

Des variables en haut référencées dans les fonctions du bas.

Un peu d'expression régulières et on obtient :

```
__ = document;
$ = "<b>Failed to load stage5</b>";
____ = 'getElementById';
_$_ = "raw";
$$ = window;
__$ = $$.crypto.subtle;
__$_ = 'decrypt';
___$ = 'status';
$____ = "importKey";
_______ = 0;
__$__ = 'then';
_$____ = 'digest';
__$___ = 'innerHTML';
___$__ = {name:'SHA-1'};
____$_ = data;
_____$ = hash;
_$_____ = Blob;
___$___ = URL;
____$__ = 'createObjectURL';
_____$ = 'type';
$_______ = 'application/octet-stream';
_$______ = 'name';
__$_____ = 'AES-CBC';
___$____ = 'iv';
____$___ = "<a href=\"";
_____$__ = "\" download=\"stage5.zip\">download stage5</a>";
_____$_ = '(';
_______$ = ')';
_________ = parseInt;
__________ = $$['navigator']["userAgent"];
_____________ = 'length';
______________ = 'substr';
_______________ = 1;
________________ = 2;
_________________ = 8;
__________________ = 16;
```

```
____________________ = 'indexOf';
_____________________ = 'charCodeAt';
______________________ = 'push';
_________________________ = Uint8Array;
____________________________ = '';
_______________________________ = 'byteLength';
________________________________ = "toString";
document['write']('<h' + 1 + '>Down' + 'load' + ' ' + 'manager</h' + 1 + '>');
document['write']("<div id=\"status\"><i>loading...</i></div>");
document['write']("<div style=\"display:none\"><a target=\"blank\" href=\"chrome://browser/content/preferences/preferences.xul\">Back to preferences</a></div>");
function MyfuncDDD(varOrParam){
  _ = [];
  for(i = 0; i < varOrParam['length']; ++i) _['push'](varOrParam['charCodeAt'](i));
  return new Uint8Array(_);
}
function toUint8Array(varOrParam){
  _ = [];
  for(i = 0; i < varOrParam['length'] / 2; ++i) _['push'](parseInt(varOrParam['substr'](i * 2, 2), 16));
  return new Uint8Array(_);
}
function MyfuncCCC(pParam){
  varOrParam = '';
  for(i = 0; i < pParam['byteLength']; ++i) {
    ___ = pParam[i]["toString"](16);
    if(___['length'] < 2) varOrParam += 0;
    varOrParam += ___;
  }
  return varOrParam;
}
function MyfuncAAA(){
  IV = MyfuncDDD(window['navigator']["userAgent"]['substr'](window['navigator']["userAgent"]['indexOf']('(') + 1, 16));
  AESKEY = MyfuncDDD(window['navigator']["userAgent"]['substr'](window['navigator']["userAgent"]['indexOf']('(') - 16, 16));
  cryptoConf = { };
  cryptoConf['name'] = 'AES-CBC';
  cryptoConf['iv'] = IV;
  cryptoConf['length'] = AESKEY['length'] * 8;
    window.crypto.subtle["importKey"]("raw", AESKEY, cryptoConf, false, ['decrypt'])['then'](function(_$___) {
    window.crypto.subtle['decrypt'](cryptoConf, _$___, toUint8Array(data))['then'](function(___$_) {
      ____$ = new Uint8Array(___$_);
      window.crypto.subtle['digest']({name:'SHA-1'}, ____$)['then'](function(____$$) {
        if(hash == MyfuncCCC(new Uint8Array(____$$))) {
          _____$_ = { };
          _____$_['type'] = 'application/octet-stream';
          hash = new Blob([____$], _____$_);
          __$____ = URL['createObjectURL'](hash);
            document['getElementById']('status')['innerHTML'] = "<a href=\"" + __$____ + "\" download=\"stage5.zip\">download stage5</a>";
```

```
        } else {
          document['getElementById']('status')['innerHTML'] = "<b>Failed to load stage5</b>";
        }
      });
    }).catch(function() {
      document['getElementById']('status')['innerHTML'] = "<b>Failed to load stage5</b>";
    });
  }).catch(function() {
    document['getElementById']('status')['innerHTML'] = "<b>Failed to load stage5</b>";
  });
}
window['setTimeout'](MyfuncAAA, 1000);
```

Pour une raison que je n'ai toujours pas compris je n'ai jamais réussi à faire fonctionner ce code dans firefox ou chrome. J'ai une erreur dans firefox qui n'est pas documentée dans l'api crypto.subtle et pas d'erreur mais pas de résultat non plus dans chrome.

C. Reverse

On fait un AES-CBC avec :

- une clé de 128 bits composée des 16 caractères du user agent précédent la première parenthèse fermante.
- Un iv composé des 16 caractères du user agent suivant la première parenthèse ouvrante.
- Le navigateur est firefox : chrome://browser/content/preferences/preferences.xul

D. Brute force de la clé/iv

En lisant la spec de firefox on trouve le format des paramètres entre les parenthèses mais comme ça a quand même l'air pénible et aléatoire sur certains device (linux et certains device Android) on se dit que les concepteurs ne sont pas des sauvages et qu'ils ont du mettre une valeur du moment.

J'ai donc essayé avec plein de valeurs trouvées sur le web : à priori toutes les valeurs représentant plus de 0,2 % du parc et bien sur la bonne n'était pas dedans.

J'ai donc récupéré le code de firefox et vérifié comment ils remplissent ce champ (ça colle à leur article sur le web, comme quoi il ne faut pas se méfier de tout).

Une fois implémenté, on trouve rapidement la réponse : « UA Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:35.0) Gecko »

V. Stage 5

A. ST20

On récupère le reference manual et les outils.

Il y a un simulateur dans ses outils. J'ai cherché un moment à faire tourner le code dessus et ne m'en suis jamais sorti.

Deuxième mauvaise piste, je pensais que input.bin était la ROM de transputer0 et cherchait donc à booter selon la procédure ROM…

une fois compris qu'on bootait en mode link, c'est devenu plus simple…

B. reverse

D'après le manuel, quand un transputer est en mode boot sur son link, il :

- interprète le permier octet (L) reçu comme la longueur des données à recevoir
- copie les L octets de données en mémoire en commençant à l'adresse la plus basse.
- Puis saute à l'adresse la plus basse de sa mémoire

```
                ; Processor       : st20c4

                ; ==================================================================

                ; Segment type: Pure code
                ; section ROM
F8                              db 0F8h                 ; length of transputer1 payload
                ; ------------------------------------------------------------------
64 B4                           ajw     -4Ch            ; adjust work space
40                              ldc     0               ; load constant

                chanIds_in_0_0:                         ; store local
D1                              stl     1
40                              ldc     0               ; load constant
D3                              stl     3               ; store local
24 F2                           mint                    ; minimum integer
24 20 50                        ldnlp   400h            ; load non-local pointer
23 FC                           gajw                    ; general adjust workspace
64 B4                           ajw     -4Ch            ; adjust work space
2C 49                           ldc     0C9h ; '+'      ; load constant
21 FB                           ldpi                    ; message to send (@ is 0x14+0xC9 --> 0xDD)
24 F2                           mint                    ; channel name (#80000000 External Channel Out 0)
48                              ldc     8               ; number of bytes to send
```

longueur

début du code

On connaît maintenant la longueur du code. On peut procéder au reverse.

On se rend compte que ça commence par écrire un message « boot OK » situé juste après un jump. On observe les messages suivants qui sont « code OK » et « decrypt ». On peut se douter que le suivant va être envoyé une fois le code de tous les transputer chargé. Il reste donc à voir comment.

On se rend compte qu'il y a une structure qui est utilisée pour indiquer au transputer la quantité de donnée à recevoir, à qui il doit les envoyer et au besoin l'offset du point d'entrée.

Quand le chargement est finit le transputer0 reçoit une structure settée à 0. Il forwarde aux transputer 1, 2 et 3 qui passent tous en mode déchiffrement.

On voit que la clé envoyée a pour valeur FF * 12, que le fichier s'appelle congratulations.tar.bz2, bref y a plus qu'à reverser tous les transputers...

une fois qu'on a recodé tous les transputers et qu'on arrive enfin à passer le vecteur de test, on se rend compte que la clé fournie ne fonctionne pas.

C. Brute force de la clé

Il faut reconnaître qu'on s'y attendait vu l'algo utilisé, la clé ne pouvait pas être la bonne. Vu qu'elle fait 12 octets, on ne peut pas envisager un brute force complet. il est nécessaire de réduire l'espace.

```
i = 0
for c in inputData:
    # set output
    output +=  chr( ord(c) ^ ( (i+2*(key[i])) % 256 ) )

    # get contributions from other transputers
    tp1 = transputer1(key)
    tp2 = transputer2(key)
    tp3 = transputer3(key)

    # update key
    key[i] = tp1 ^ tp2 ^ tp3
    # update i
    i = (i+1) % 12
```

En observant la fonction de déchiffrement on voit que la sortie est quasi directement xorée avec la clé. On peut donc considérer une attaque en clair connu. Pour le clair comme on sait que c'est un tar.bz2, on va utiliser l'header bzip2.

```
.magic:16                  = 'BZ' signature/magic number
.version:8                 = 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1 (deprecated)
.hundred_k_blocksize:8     = '1'..'9' block-size 100 kB-900 kB (uncompressed)

.compressed_magic:48       = 0x314159265359 (BCD (pi))
.crc:32                    = checksum for this block
.randomised:1              = 0=>normal, 1=>randomised (deprecated)
.origPtr:24                = starting pointer into BWT for after untransform
```

Si on suppose que .hundred_k_blocksize vaut 9 ce qui a l'air d'être la valeur par défaut, on connaît 10 octets. Pour les 2 derniers qui sont du crc on ne sait à priori rien.

Pour chaque octet connu, deux octets de la clé peuvent correspondre. En effet le xor est fait avec 2*(key[i])) qui fait que le bit de poids fort de la clé n'est pas utilisé.

On se retrouve donc avec un espace de clé de 2exp10*256exp2 = 6,7 M de clé.

C'est jouable mais par contre pas en python. Une fois recodé en C et multithreadé je suis encore loin du compte :sur ma machine il me faudrait environ 10 jours (sachant qu'en plus le code est surement bogué…)

Il faut donc trouver des moyens de couper des branches au plus vite.

En regardant l'header de plus près on voit que :

- .randomised doit valoir 1 et c'est un bit de poid fort (donc bien dépendant de la clé) on peut déjà stopper la moitié des tests (plus que 5 jours…).

- .origPtr est une valeur sur 24 bits qui présente un offset. Je n'ai pas trop cherché ce qu'est cete offset mais à priori c'est l'adresse des données une fois la première passe de compression effectuée. Intuitivement je me suis dit qu'il était peu probable que l'offset soit supérieur à la longueur du fichier compressé (ratio de 2).

Comme le fichier fait dans les 250K, on divise le temps de calcul par 2exp24 / 250000 = 67. On tombe à moins de 2heures.

Les hypothèses étaient bonnes, la clé est [0x5E, 0xD4, 0x9B, 0x71, 0x56, 0xFC, 0xE4, 0x7D, 0xE9, 0x76, 0xDA, 0xC5]

VI. Stage 6

On se retrouve avec un jpeg appelé congratulations.jpg. On est arrivé à la fin…

et non en fait.

Avec binwalk on voit qu'il y a un tbz de caché à la fin de l'image. On récupère la fin et le décompresse. On obtient un png :

avec pngchunkks on voit qu'il y a des chunks nommés sTic :

```
Chunk: Data Length 13 (max 2147483647), Type 1380206665 [IHDR]
  Critical, public, PNG 1.2 compliant, unsafe to copy
  IHDR Width: 636
  IHDR Height: 474
  IHDR Bitdepth: 8
  IHDR Colortype: 6
  IHDR Compression: 0
  IHDR Filter: 0
  IHDR Interlace: 0
  IHDR Compression algorithm is Deflate
  IHDR Filter method is type zero (None, Sub, Up, Average, Paeth)
  IHDR Interlacing is disabled
  Chunk CRC: -1707043784
[...]
Chunk: Data Length 4919 (max 2147483647), Type 1667847283 [sTic]
  Ancillary, public, in reserved chunk space, safe to copy
  ... Unknown chunk type
  Chunk CRC: -2040313934
[...]
Chunk: Data Length 4919 (max 2147483647), Type 1667847283 [sTic]
  Ancillary, public, in reserved chunk space, safe to copy
  ... Unknown chunk type
```

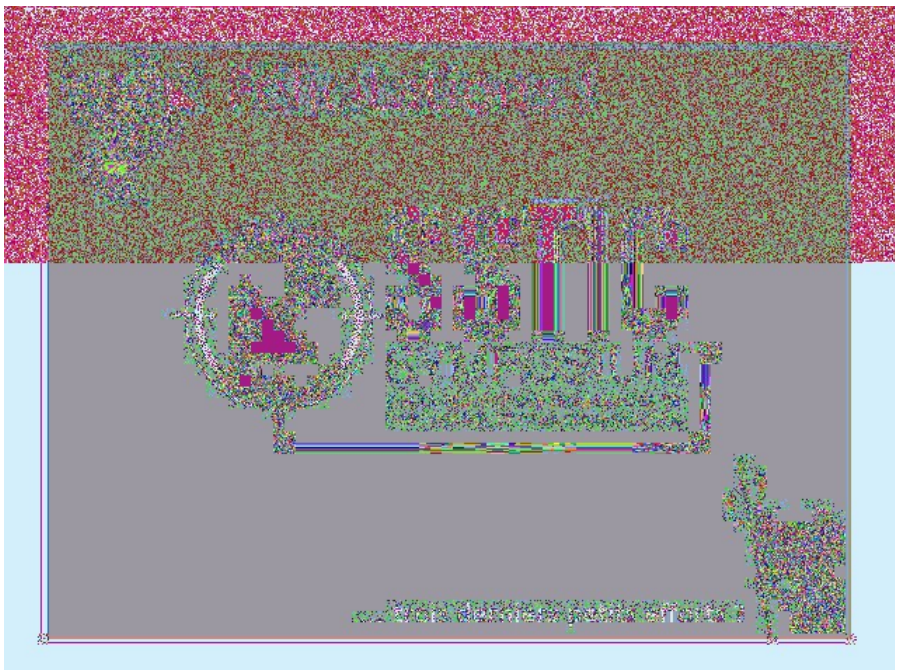
Après extraction et décompression on obtient un tiff :

Là ça a été beaucoup plus long... Je pensais qu'on avait à peu près la même chose qu'au stage précédent mais en analysant le format de fichier on s'aperçoit que ce sont des données décompressées et vu la taille du fichier il n'y a pas de données complémentaires cachées dedans.

J'ai ensuite pensé qu'il y avait une couleur très proche d'une autre (genre du noir ou du blanc) qui faisait qu'on ne voyait pas le texte caché dedans. J'ai donc regardé la valeur de chaque pixel et sa fréquence. On arrive à :

```
0x000000, 37906
0x000001, 6801
[...]
0x000101, 7147
[...]
0x000100, 6711
[...]
0x010101, 1168
[...]
0xa56828, 2080
[...]
0xfffefe, 16830
0xfffeff, 17786
[...]
0xffffe, 17898
0xffffff, 132006
```

Sur les blanc la magouille semble évidente. En remplaçant les couleurs par des couleurs tirées au sort. On obtient :



Les pixels modifiés on l'air d'être au début de l'image. Comme il reste quand même pas mal de combinaisons possible pour reformer l'image et qu'on ne sait même pas quel type de fichier on est sensé obtenir j'ai tenté toutes les solutions de stéganographie disponibles sur le web. Une bonne perte de temps aucune n'a donné quelque chose.

En observant la fréquence des valeurs on voit 3 valeurs autours de 17K : **0xfffefe, 0xfffeff, 0xffffe.** On peut donc raisonnablement penser que seuls le bit de poids faible de 2 couleurs est utilisé. On construit donc un fichier avec le dernier bit de ces deux couleurs et on teste le résultat avec binwalk. On tente plusieurs combinaisons (bit de poids faible, bits de poids fort, …) et on trouve finalement assez vite un tbz.

Cette fois on obtient un gif :

J'ai commencé par regarder les couleurs présentes dans la palette gif. Il y avait beaucoup de noir…

En changeant le bon noir de la palette on obtient ça :